DOCUMENT RESUME

ED 232 676 IR 050 320

AUTHOR Lofstrom, Mats
TITLE A Description of the Text Data Base System TDBS. Stockholm Papers in Library and Information Science.
INSTITUTION Royal Inst. of Tech., Stockholm (Sweden). Library.
SPONS AGENCY National Swedish Board for Technical Development, Stockholm.
REPORT NO TRITA-LIB-4077
PUB DATE Nov 82
NOTE 28p.; The project was also funded by the Delegation for Scientific and Technical Information. The report may be referred to as Technical Report 822, Paralog, November 1982.
PUB TYPE Reports - Descriptive (141)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS Check Lists; *Computer Programs; *Databases; Foreign Countries; *Information Retrieval; Information Storage; *Online Systems; Specifications
IDENTIFIERS *Database Management Systems; Sweden; *Text Handling

ABSTRACT

Because experience with large information retrieval (IR) and database management (DBM) systems has shown that they are not adequate for the handling of textual material, two Swedish companies--Paralog and AU-System Network--have joined in a venture to develop a software package which combines features from IR and DMB systems to form a Text Data Base System (TDBS) with text retrieval functions, structuring and immediate database updating capabilities, and a well-defined interface between the DBM system and an application program. The first version of TDBS will be implemented using a portable subset of Pascal on a VAX 11/780 and will be available during 1983. In this preliminary report, TDBS is specified as using the package concept of the Ada language. Following a list of TDBS features, the report describes: (1) TDBS data structure, with definitions of field types, composite types, databases, and an example of how to use the structuring facilities of TDBS; (2) TDBS data manipulation, including file handling, searching, and structure handling; (3) updating of TDBS files; and (4) the interface between TDBS and an application program. A list of eight references and an appendix defining the eight Ada packages which are used to specify TDBS conclude the report. (ESR)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED232676

REPORT TRITA-LIB-4077 NOVEMBER 1982

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☒ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

A DESCRIPTION OF THE TEXT DATA BASE SYSTEM TDBS

Mats Löfström

Paralog AB
Stockholm, Sweden

IR050320

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

S. Schwarz

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

ESSEP

Stockholm Papers in Library and Information Science
TRITA-LIB-4077, RITL, Stockholm 1982
ISSN 0346-9042

The report can also be referred to as Technical Report 822, Paralog, November 1982.

This project is sponsored by the Swedish National Board for Technical Development (STU) and The Delegation for Scientific and Technical Information (DFI).

© Paralog AB, Stockholm 1982

TABLE OF CONTENTS

INTRODUCTION 5

DATA STRUCTURE 7
- Field types 8
- Composite types 9
- Database 10
- An example 11

DATA MANIPULATION 14
- File handling 14
- Searching 14
- Structure handling 17

UPDATING OF TDBS FILES 18

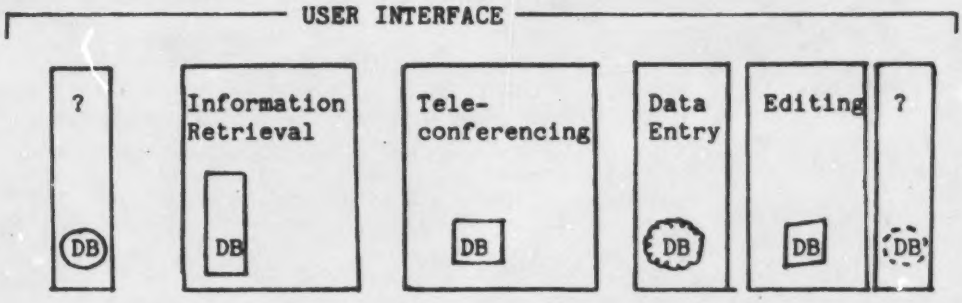
INTERFACE TO AN APPLICATION PROGRAM 19

REFERENCES 20

APPENDIX 21

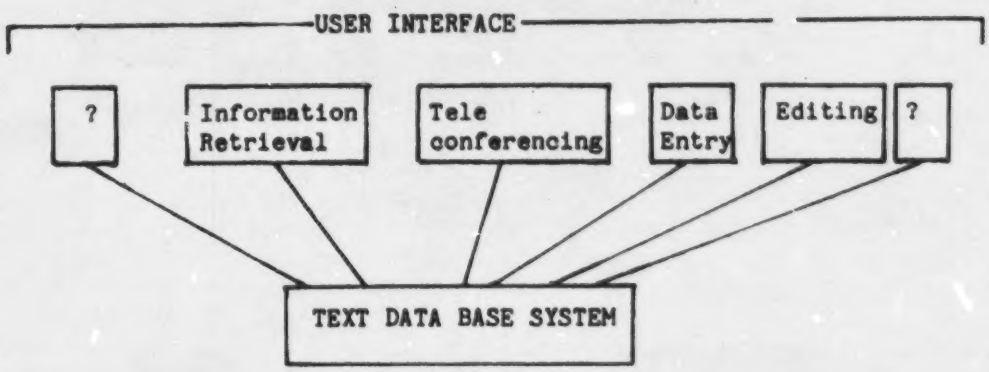
INTRODUCTION

Experiences from the design, implementation and use of one of the large information retrieval (IR) systems 3RIP <1>, and the use of many of the common data base management (DBM) systems <2>, has shown that these system types are not adequate tools for building integrated systems for the handling of textual material <3>. Such systems could include functions for information retrieval, teleconferencing, data entry, editing and so on in a configuration like the one below:



DB stands for the part of each subsystem used for the storage and retrieval of data. It is noticeable, that although all subsystems handle mostly textual information of similar type, they use different techniques to handle this information. In the case of the IR and teleconferencing subsystems they normally use their own database systems, while the other may use the filehandling system provided by the host machine on which the system is implemented. This fact severly reduces the possibility to make a smooth integration of the subsystem, and the system designer is bound to build his own data base system, or try to convert between the existing data base structures, both solutions being time consuming and in most cases leading to a system that is unreliable, and to which it is difficult to add new functions.

One solution to this problem is to provide a data base system especially designed for the handling of textual material, that could be used for the building of such systems in a manner shown below:



The need for such a system has been the starting point of the Text Data Base System (TDBS) project, a joint venture of the Swedish companies Paralog and AU-System Network, aimed at developing a software package combining text retrieval functions found in IR-systems with useful features from DBM- systems, e.g. structuring capabilities, the possibility to perform immediate updates of the data bases and above all the existence of a well defined interface between the DBM-system and an application program. When implemented TDBS will be a system having:

1. a high capacity, i.e many users (application programs) can search large TDBS data bases simultaneously, getting acceptable response times,

2. few limits on the file sizes, record lengths, types of characters etc. that an application program can use,

3. structuring capabilities adequate for the representation of large texts. This includes primitives for dividing texts into paragraphs, sentences and words but also general facilities for associating text parts into tree and/or network structures,

4. facilities for mixing textual material with numerical and/or binary information,

5. very powerful text retrieval functions, including the searching of combinations of individual words or part of words and also facilities, e.g thesaurus structures, to represent semantic relationships between searchable words or word stems,

6. file handling facilities, that can be used freely by an application program to implement for instance user dialogues, help facilities, error messages and macro functions,

7. an interface that is easy to use both for the applications programmer and the data base administrator,

8. reliable security function including transaction logging and recovery from system failures,

9. the possibility to perform immediate updates of the files of the system.

TDBS will be specified (as shown in this report) using the package concept of the Ada language <4>. The first version of the system will be implemented using a portable subset of Pascal on a VAX 11/780 and will be available during 1983.

The specification given below is preliminary and incomplete. Nothing is said, for instance, about the simultaneous updates of the files by several application programs, functions for recovery from system failures and the facilities for the data base administrator to place the files on various physical devices. Changes to the specifications are expected when TDBS has been tested in a number of applications. Having completed and made necessary revisions to the prototype version of TDBS, an implementation of the entire system using Ada is planned.

DATA STRUCTURE

In the description of TDBS we make the distinction between the definition of a TDBS type, e.g a record, and an occurrence of the type within a TDBS database. A type is defined using a type descriptor (similar to what is found in Data Definition Languages (DDL) of DBM-systems), while occurrences of the type are created (and possibly stored in a database) and retrieved using the operations defined for the particular type.

A TDBS type is described by its set of values and the operations defined for the type. The most elemental type of TDBS is the field. A record is a named group of one or more associated fields, each occurrence of a record having a unique name. Records of the same type are stored in files. A structure in TDBS defines a logical structure used to associate record occurrences. A structure might be viewed as a list, the elements being record occurrences. In the structure descriptor it is possible to define the record type allowed in the particular structure and also if record occurrences are to be ordered in any way when inserted into the structure. It is also possible to define an owner of the structure to be an occurrence of a record of a certain type.

Finally, a database in TDBS consists of one or more files possibly associated with a number of structures.

FIELD TYPES

The following field types are recognized within TDBS:

1. TDB_Character

   The values of this type defines the basic character set of an implementation of TDBS. In the first version of TDBS this set will be based on the Teletex standard <5>.

2. TDB_String

   A arbitrarily long sequence of TDB_Character.

3. Text

   An arbitrarily long sequence of TDB_Character.
   This type differs from TDB_String in that all words of an occurrence of the field are retrievable using the search operations defined for the type Text (see Data Manipulation below).

4. Date

   Date is a TDB_String having a number of recognized forms for representing a date.

5. Time

   A TDB_String with a number of recognized forms for representing time.

6. TDB_Integer

   All integers between an upper and a lower limit, depending on the programming language or the host machine being used.

7. TDB_Real

   Real numbers in a floating point representation.

8. Bitstring

   An arbitrarily long sequence of bits.

9. TDB_Enumeration

   A field of this type is defined by enumerating all its possible values. A field defined as TDB_Enumeration(monday,tuesday) can only have the values 'monday' or 'tuesday'.

COMPOSITE TYPES

The following composite types are used to define a database:

1. List

   An arbitrarily long sequence of occurrences of a certain type. The definition of a list type Ltype1 with Bitstring as elements of the list has the form:

```
Ltype1 IS List of Bitstring
```

2. Record

   A record type is composed by a number of the above types. The definition of a record type Rtype having N fields is

```
Rtype IS Record
        Field1;
        Field2;
         ......
        FieldN;
        End Record;
```

   Each occurrence of a record is identified by a name, which must be unique within a database.

3. File

   A file is a "List of Record", or if the Record has only one field, a file is simply "List of Field".

   A number of operations for the file type are defined in the Appendix.

4. Structure

   A structure type defines possible associations between record occurrences. The definition of a simple structure Struc is:

```
Struc IS Structure of (Rtype1,Rtype2);
```

   Rtype1 and Rtype2 are previously defined record types. An occurrence of Struc will have one occurrence of Rtype1 as its owner and one or more occurrences of Rtype2 as its members. (Note here that record occurrences are always stored in files. A structure occurrence, then, consists of references to the record occurrences included in the particular structure.)

It is allowed for a record occurrence to be an owner and a member of different structure occurrences. A record occurrence must not, however, be present more than once in any one structure occurrence.

Each occurrence of a structure is identified by a name, unique within the database.

In the Appendix a number of operations on a structure are defined.

DATABASE

A definition of a database comprises a number of record, file and structure definitions. The database New is defined as:

```
New IS Database

        <Record definitions>;

        <File definitions>;

        <Structure definitions>;

        End Database;
```

To each occurrence of a database an important attribute called Open_Files is associated, giving information on what files or parts of the files are currently open for searching. The operations on databases are defined in the Appendix.

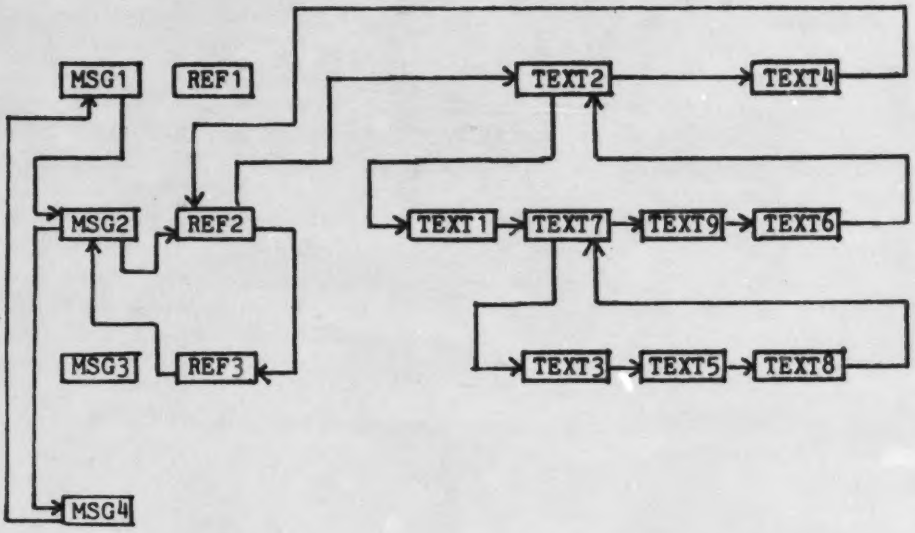
AN EXAMPLE

As an example of how to use the structuring facilities of TDBS and how to define a database, consider a system for the handling of simple messages, each message consisting of a receiver, a sender, a date and the message. It should be possible not only to store and retrieve the messages but also to let a number of messages be connected to one message forming what could be called "comment chains". Moreover a message can be associated to a list of literature references, each reference having one or more authors, a publication date, keywords and an abstract of the text. It should also be possible to store the full text of a document being referenced and connect this representation to the reference. If the text of the document consists of several text parts, structured as a list or a tree, it must be possible to represent this fact within the database.

A database definition for such a system could look like:

```
Message IS Database

        Msg IS Record
                Receiver :  Text;
                Sender :  Text;
                Date :  Date;
                Text :  Text;           -- The message
                End Record;

        Reference IS Record
                Author :  List of Text;
                Date :  Date;           -- The publication date
                Keywords :  List of Text;
                Text :  Text;           -- The abstract
                End Record;

        Textpart IS Record
                Text :  Text;           -- The full text
                End Record;

        Msg_File IS File of Msg;

        Ref_File IS File of Reference;

        Text_File IS File of Textpart;

        Msg_Msg IS Structure of (Msg,Msg);

        Msg_Ref IS Structure of (Msg,Reference);

        Ref_Text IS Structure of (Reference,Textpart);

        Text_Text IS Structure of (Textpart,Textpart);

End Database;
```

A snapshot of a small part of this database could look like:



MSGi, REFi and TEXTi are the identifiers of occurrences of the type Msg, Reference and Textpart, respectively.

The following structure occurrences exist (the owner of the structure occurrence is mentioned above it's members):

Occurrence	Type
MSG1	Msg_Msg
MSG2, MSG4	
MSG2	Msg_Ref
REF2, REF3	
REF2	Ref_Text
TEXT2, TEXT4	
TEXT2	Text_Text
TEXT1, TEXT7, TEXT9, TEXT6	
TEXT7	Text_Text
TEXT3, TEXT5, TEXT8	

A few comments could be made:

1. If it had sufficed to represent the structure of the text of a document in a two level tree structure, it would have been enough to associate the field type Text to the Reference record. Besides imposing restrictions on the number of levels that could be represented within the structure, this representation would mean that the entire document is retrieved when searching, leading to problems when the document is very large. The advantage, of course, is that the database will be less complex, since three types, (Textpart, Ref_Text and Text_Text) are being removed.

2. It is possible for the applications programmer to handle the "comment chains" without using the structuring facilities, by introducing a type Comments defined as "List of TDB_String" into the Msg record definition. The identifier of each comment to a certain message is then stored as an entry in the Comment field of this particular message occurrence. This solution reduces the database complexity, but may introduce errors in the database, since no checking of types are performed by TDBS when inserting an identifier into the comment chain.

The above notes show that there are several options open for setting up a database for a certain application. The decisions are among other things guided by the need for:

- fast updating of the files,
- flexible structuring facilities,
- a high level of security when handling structures and
- small amounts of text as the result of searching the files

in the particular application.

DATA MANIPULATION

Data in TDBS is handled using a number of procedures and functions defined in the Appendix. Below follows a short description of some important features.

FILE HANDLING

A TDBS record is stored on a file and can be retrieved (brought into memory) using the identifier of the record as one of the arguments to the procedure Get. For each file there is an alphabetically sorted directory of all its record identifiers, that can be consulted using the Directory procedure. The procedures Create, Put, Delete, Rename and Protect are also defined for a record.

Parameters are passed between a field of a particular record and variables in the application program using the procedures Getvalue and Putvalue.

Example:
Change the value of the field named Date of the message M1012 to "1980-10-20".

```
Getrec(MSGFILE; M1012; P; Current_Record_In_File);
```
(P points to an internal representation of the record. MSGFILE is the name of an occurrence of the type Msg_File. Current_Record_In_File is irrelevant here).

```
Putvalue("Date"; "1980-10-20"; P);
```
(Change the value).

```
Put(MSGFILE; M1012; P);
```
(Write the message back onto the file).

SEARCHING

The basic retrieval facility of TDBS is the searching of the entire database using the Search procedure, with which all files can be searched in parallell. The domain of the database to be searched by the procedure depends on:

- what files and part of files are open for searching at the time of the procedure call. The procedures Open and Close are used to change this domain.
- what field(s) of the records of the database the search is to be performed in.

Example:
Assume, that all files of the Message database are open. The procedure calls

```
Close(REFFILE; -1; -1; Open_Files);

Close(MSGFILE; -1; -1; Open_Files);

Open(MSGFILE; M1000; M2000; Open_Files);
```

will close all parts of the files REFFILE and MSGFILE (REFFILE is an occurrence of a file of type Ref_File) except for the records of MSGFILE, having identifiers between M1000 and M2000. This fact will be represented by the value of Open_Files.

A procedure call

```
Search(Open_Files; "Date"; BETWEEN; "1980-01-01";
       "1980-12-31"; SP);
```

following the above sequence of procedure calls, will retrieve all messages with identifiers between M1000 and M2000, being sent during 1980.

Searching is performed using the Search procedure, which finds all record occurrences satisfying the given search criteria. The latter is expressed using the form:

Fieldname	Operator	Value

Example:

"Date"	EQUALS	1980-10-22

The Fieldname is the logical name defined for a field within a certain record type of the database.

The operator defines how the fieldname and the value(s) must be related if a record occurrence is to satisfy the search criteria. Different operators are defined for the various field types:

1. For Text

   Operator: CONTAINS

   Value: A TDB_String

   When fields of the above type are made retrievable (by using the Change or Modify procedures) a number of retrieval strings are created. Such a retrieval string is usually a word and we refer to it as a word below. So the TDB_String can be a word, in which case the Search procedure finds all records having one or more occurrences of the word in the given field.

The value can also include so called "pattern matching characters", indicating that the search is to be for a certain pattern that is found within words.

Example:
Search(...; "Text"; CONTAINS; oxi; ....);

will retrieve records where one or more words of the field named "Text" include the given pattern. Example of such words are "proximity" and "oxides".

2. For all other field types except TDB_Character and Bitstring.

Operators: EQUAL, LESS, LEQ, GREATER, GEQ and BETWEEN.

Value: A value of the appropriate type.
(The operator BETWEEN takes two values).

The result of the Search procedure is a set of all record occurrences satisfying the search criteria. All sets created can be saved and later combined using the Combine procedure, forming new sets.

Such a combination of sets can be expressed as:

SP1 Operator SP2

where SP1 and SP2 are pointers to internal representations of sets.

Example:
Assume that all files of the Message database are open. The procedure calls

```
Search(Open_Files; "Date"; BETWEEN; "1981-01-01";
       "1981-12-31"; SP1);

Search(Open_Files; "Text"; CONTAINS; "train"; SP2);

Combine(SP1; SP2; AND; SP3);
```

will create a set of all messages and references, sent (published) during 1981, having the word "train" in their textual part.

For all sets the operators AND, OR and ANDNOT are defined. If two sets are the result of searches in fields of the type Text (as SP2 above), other operators are also available to combine these sets. The result of the combination can then be restricted to record occurrences, where any two words must occur within a certain distance (measured as a number of words). The operator has the form Wn (A Wn B means that A and B must occur within n words of each other if a record occurrence is to retrieved).

Example:

Search(Open_Files; "Text"; CONTAINS; "bicycle"; -1; SP4);

Combine(SP3; SP4; W10; SP5);

This results in a set of messages and references, sent (published) during 1981, having the words "train" and "bicycle" within ten words.

Finally, it should be mentioned that all retrievable words are collected in the vocabulary of the database, where the words are alphabetically sorted. The procedure Vocabulary is used to examine the vocabulary, and procedures will also be defined, with which the retrievable words can be structured for instance into a thesaurus.

STRUCTURE HANDLING

Procedures exist to create, store and retrieve structures using their structure identifier. As for files all structure identifiers of a database are alphabetically ordered and can be examined using the Dir_Structure procedure. It is possible to search through all structures for the occurrences having a certain record as a member or an owner. The procedure Search will create a set of all such structures, which can then be further examined.

Example:
We want to find the identifier of the first member of one structure owned by the first record occurrence of the set SP5 above.

Reference(Current_Reference; R);
(Assigns to R the identifier of the first record of the set SP5).

Search(R; MEMBER; Current_Structure; SEP);
(Searches for all structures having the above record as a member. SEP points to an internal representation of a list of references to all structure occurrences satisfying the search criteria).

Get_Structure(SEP; CURRENT; Current_Structure;
Current_Reference; SP);
(Brings into memory the first structure occurrence of the above set).

Get_Owner(SP; RES);
(The variable RES now has the wanted identifier as its value).

UPDATING OF TDBS FILES

The execution of the file handling procedures Get, Put, Rename, Delete and Protect immediately updates the file involved. If the Put procedure is used to replace an existing record occurrence, then the old version is also available for backup purposes.

The structure handling procedures also make the newly added or modified structures immediately retrievable, using either the Get_Structure or the Search procedure.

If we want the contents of a new or added record occurrence to be retrievable by the Search procedure for records, the procedure Change is to be used instead of Put.

Finally, there is the procedure Modify, taking a database as its input. This procedure performs all updates necessary, due to the addition, deletion or modification of record occurrences within the entire database, since the last call of Modify. After the execution of Modify the contents of all files is retrievable using the Search procedures.

INTERFACE TO AN APPLICATION PROGRAM

The interface between TDBS and an application program is given in the Appendix as a number of Ada packages, defining what is equivalent to a Data Manipulation Language of other DBM systems. One of the packages (called TDB_Define) describes an external representation of a database that TDBS can understand, thus giving a sort of Data Definition Language for TDBS.

The advantages of using Ada as the specification language are among other things (see <6> and <7>) that this technique makes it possible to:

1. indicate exactly what types are available to the applications programmer together with all operations defined for these types. By using a package for one or more such types it is possible to implement so called "abstract data types" <8>. So is, for instance, an implementation of the abstract data type Search_se included in the TDB_Search package. The internal representation of a Search_set (holding the result of a search operation) is unknown to the applications programmer and the type can only be manipulated using the procedures defined within the package. (These procedures can be seen as implementations of the abstract definitions of the operations available on the data type Search_set).

2. write application programs using Ada without fully implementing TDBS. The specification of TDBS can thus be refined stepwise, e.g. when the system does not meet the requirements of an important application, the specification of the entire system can be consulted to find out if changes can be made.

3. to build subsystems of TDBS by combining some but not all of the packages. Thus a system for file handling only, is constructed by the packages TDB_Names, TDB_Define, TDB_Record and TDB_File.

In the first version of TDBS, written in Pascal, an interface to this language will be provided. Later versions will have interfaces to other high level languages including Ada. For the latter language the objective is to be able to implement the TDB_Define package in such a way that databases can be defined using Ada constructs only, thus creating a TDBS system that is "tightly coupled" (see <2>) to the programming language Ada.

REFERENCES

1. 3RIP: An Introduction. Paralog, Stockholm, 1982.

2. Date C.J. An Introduction to Database Systems. Addison-Wesley, 1981.

3. Löfström M. Informationssökningssystem - Problem och möjligheter. I konferensdokumentation för Norddata-82. (In Swedish).

4. Reference Manual of the Ada Programming Language. Department of Defense, Washington D.C. , November 1980.

5. Draft Recommendations for an International Teletex Service. Televerket Teletex, UDP 80017. Stockholm, November 1980.

6. Bowles K. Linked Ada Modules Shape Software Systems. Electronic Design, July 22, 1982, pp 117-126.

7. Meyer B. Principles Of Package Design. CACM, July 1982, pp 419-428.

8. Baroody J. and DeWitt D. An Object-Oriented Approach to Database System Implementation. Transactions on Database Systems, Vol. 6, Nr. 4, pp 576-601.

APPENDIX

The following Ada packages are used to specify TDBS:

1. TDB_Names
2. TDB_Database
3. TDB_Define
4. TDB_File
5. TDB_Record
6. TDB_Structure
7. TDB_Search
8. TDB_Thesaurus

```
PACKAGE TDB_Names IS

        TYPE
        String100 IS ARRAY(INTEGER RANGE 0..100) OF CHARACTER;
        Identifier IS ARRAY(INTEGER RANGE 0..32) OF CHARACTER;
        Database_id;  File_id, Record_id, Structure_id IS Identifier;
        Protection_Code IS PRIVATE;
        Inspect_Mode IS (Current,First,Last,Next,Previous);
        Index_Entry IS  RECORD
                        -- Structure to be defined
                        END RECORD;
        Index_Structure IS ARRAY(INTEGER RANGE <>) OF Index_Entry;

END TDB_Names;
```

```
PACKAGE TDB_Database IS

        USE TDB_Names;  USE TDB_File;  USE TDB_Structure;
        USE TDB_Search;

        PROCEDURE
        Open(D:  Database_id;  OF:  INOUT Open_Files);

        Close(D:  Database_id;  OF:  INOUT Open_files);

        Create(D:  Database_id;  DD:  Database_Descriptor);

        Modify(OF:  Open_Files);
        -- Takes as input all record occurrences that have been changed or
        -- added to the file since the last call of Modify.  The procedure
        -- performs all necessary updates needed for forthcoming searches.

        Change(R:  Record_id;  F:  File_id;  P:  Record_Pointer);
        -- Writes the record occurrence (pointed at by P) onto a file and
        -- performs all the updates resulting from the change (or addition)
        -- of this record occurrence making it
        -- immediately retrievable using the Search procedure.

        Add_File(D:  Database_id;  F:  File_id;  OF:  INOUT Open_Files);

        Delete_File(D:  Database_id;  F:  File_id;  OF:  INOUT Open_Files);

        Dir_File(D:  Database_id;  DR:  Directory);

        Dir_Structure(D:  Database_id;  DR:  Directory);

        -- Exceptions

END TDB_Database;
```

```
PACKAGE TDB_Define IS

        USE TDB_Names;

        TYPE
        -- External_Character_Set_Descriptor IS to be defined
        -- External_File_Descriptor IS to be defined
        -- External_Structure_Descriptor IS to be defined
        -- External_Database_Descriptor IS to be defined
        Character_Set_Descriptor IS LIMITED PRIVATE;
        File_Descriptor IS LIMITED PRIVATE;
        Structure_Descriptor IS LIMITED PRIVATE;
        Database_Descriptor IS LIMITED PRIVATE;

        PROCEDURE
        -- For each of the above types, there are two procedures:

        Define(EFD:  External_File_Descriptor;  FD:  OUT File_Descriptor);
        -- Parameters:
        -- EFD:  The value of a variable declared in the applications
        -- program to be of type External_File_Descriptor

        Show(FD:  File_Descriptor;  EFD:  OUT External_File_Descriptor);

        -- Exceptions

END TDB_Define;
```

```
PACKAGE TDB_File IS

        USE TDB_Names;  USE TDB_Record;

        TYPE
        Current_Record_In_File IS LIMITED PRIVATE;
        Current_File IS LIMITED PRIVATE;
        Directory IS Index_Structure;

        FUNCTION
        Current_Rif RETURN Current_Record_In_File;

        Current_File RETURN Current_File;

        PROCEDURE
        Set_Rif(R:  Record_Id;  C:  INOUT Current_Record_In_File);

        Set_File(F:  File_id;  C:  INOUT Current_File);

        Getrec(F:  File_id;  R:  Record_id;  P:  OUT Record_Pointer;
               C:  INOUT Current_Record_In_File);

        -- Loads a TDB_Record into memory and points to it using
        -- Record_Pointer
        -- Parameters:
        -- Default for F is the value of Current_File

        Get(F:  File_Id;  C:  INOUT Current_Record_In_File;
               I:  Inspect_Mode;  P:  OUT Record_Pointer);

        Put(F:  File_id;  R:  Record_id;  P:  Record_Pointer);

        Protect_File(F:  File_id;  P:  Protection_Code);

        Protect_Rec(F:  File_id;  R:  Record_id;  P:  Protection_Code);

        Delete_Rec(F:  File_id;  R:  Record_id);

        Rename_Rec(F:  File_id;  From,To:  Record_id);
        -- Put, Protect, Delete and Rename all update the directory of F

        Delete_File(F:  File_id);

        Rename_File(From,To:  File_id);

        Copy_Rec(Fromrec:  Record_id;  From_File:  File_Id;
                 Torec:  Record_id;  To_File:  File_id);

        Copy_File(From,To:  File_id);

        Create_File(F:  File_id;  FD:  File_Descriptor);

        Directory(F:  File_id;  From,To:  Record_id;  D:  OUT Directory);

        -- Exceptions

END TDB_File;
```

```
PACKAGE TDB_Record IS

USE TDB_Define; USE TDB_Names;

        TYPE
        -- TDB_Character IS to be defined
        TDB_String IS ARRAY(INTEGER RANGE < >) OF TDB_Character;
        Text IS ARRAY(INTEGER RANGE <>) OF TDB_Character;
        TDB_Integer IS INTEGER;
        TDB_Real IS REAL;
        -- Date IS to be defined
        -- Time IS to be defined
        -- Bitstring IS to be defined
        -- TDB_Enumeration IS to be defined
        -- List IS to be defined
        TDB_Record IS LIMITED PRIVATE;
        Record_Pointer IS PRIVATE;

        -- For each of the above field types
        -- two procedures GETVALUE and PUTVALUE
        -- exist, tranforming the representation of data of the given
        -- type between the internal representation
        -- (found in the record occurrence pointed at by
        -- Record_Pointer) and a variable in the application program.
        -- These procedures all have the same syntax as those given
        -- below for the field type Text.

        Getvalue(Name:  String100;  T:  OUT Text;  RP:  Record_Pointer);

        -- Parameters:
        -- Name:  The logical name (given by the external file descriptor)
        --        of a field of type Text in the record
        --        occurrence pointed at by RP
        -- T:  A variable declared in the applications program to be of
        --     type Text

        Putvalue(Name:  String100;  T:  Text;  RP:  Record_Pointer);

        Create_Rec(R:  Record_id;  FD:  File_Descriptor;  RP:  OUT
                       Record_Pointer);

        Rename_Rec(From,To:  Record_id);

        Delete_Rec(R:  Record_id);

        -- Exceptions

END TDB_Record
```

```
USE TDB_Define; USE TDB_Names;

        TYPE
        Search_Pointer IS LIMITED PRIVATE;
        Current_Structure IS LIMITED PRIVATE;
        Structure_Pointer IS LIMITED PRIVATE;
        Current_Reference IS LIMITED PRIVATE;
        Search_Mode IS (OWNER,MEMBER);

        FUNCTION
        Father(C:  Current_Reference) RETURN BOOLEAN;
        -- Tests whether the record occurrence identified by Current_Reference
        -- is the owner of other structure occurrences.

        PROCEDURE
        Search(R:  Record_id;  SM:  Search_Mode;
               CS:  INOUT Current_Structure;  SP:  OUT Search_Pointer);
        -- Searches all structures and creates a set of all structure
        -- occurrences satisfying te search criteria, i.e.  that R is
        -- either the owner of (SM is then equal to OWNER) or the
        -- member of a structure occurrence.  The set created is pointed at
        -- by SP.  Current_Structure is assigned the value of the identifier
        -- of the first structure occurrence of this set.
        -- Sideeffects:
        -- If Search_Mode is MEMBER, the member with it's identifier equal to
        -- R of each structure occurrence in the set is marked
        -- as the one satisfying the search criteria

        Structure(CR:  Current_Structure;  S:  OUT Structure_id);

        Get_Structure(S:  Search_Pointer;  G:  Inspect_Mode;
               CS:  INOUT Current_Structure;  CR:  INOUT Current_Reference;
               SP:  OUT Structure_Pointer);
        -- Brings a structure occurrence from S into memory and points
        -- to it using SP.  The structure to chose is governed by the
        -- values of S and G.  Current_Structure is set to the structure
        -- chosen.  The procedure sets Current_Reference to the first
        -- member of the chosen structure, or, if a member has been marked
        -- by the Search procedure (see Search) to the marked member.

        Get_Owner(S:  Structure_Pointer;  R:  Record_id);

        Get_Reference(S:  Structure_Pointer;  CR:  INOUT Current_Reference;
                      G:  Inspect_Mode;  R:  Record_id);

        Change_Owner(SP:  Structure_Pointer;  From,To:  Record_id);

        Insert_Reference(SP:  Structure_Pointer;  G:  Inspect_Mode;
                      R:  Record_id);

        Delete_Reference(SP:  Structure_Pointer;  G:  Inspect_Mode);

        Create_Structure(SP:  OUT Structure_Pointer;SD:  Structure_Descriptor;
                      O:  Record_id);
        -- Creates a structure, with the owner O, and points to using
        -- SP.

        Put_Structure(SP:  Structure_pointer);

        Get_Structure(SP:  OUT Structure_Pointer;  S:  Structure_id);

        Delete_Structure(S:  Structure_id);

END TDB_Structure;
```

```
PACKAGE TDB_Search IS

USE TDB_Record; USE TDB_Names;

        TYPE
        Open_Files IS ARRAY(INTEGER RANGE 0..MAXFILES) OF File_id;
        Search_Pointer IS PRIVATE;
        Vocabulary IS Index_Structure;
        Current_Reference IS LIMITED PRIVATE
        -- Combination_Modes IS to be defined
        Operators IS (CONTAINS, EQUAL, LESS_OR_EQUAL, GREATER_OR_EQUAL,
                      GREATER_THAN, LESS_THAN, BETWEEN);

        PROCEDURE
        Open(F:  File_id;  From,To:  Record_id:  OF:  INOUT Open_Files);

        Close(F:  File_id;  From,To:  Record_id;  OF:  INOUT Open_Files);

        Search(F:  Open_Files;  T:  String100;  O:  Operator;
               Value1,Value2:  TDB_String;  CR:  INOUT Current_Reference;
               SP:  OUT Search_Pointer);

        -- Searches all open files and creates a set of
        -- references to all record occurrences satisfying the
        -- search criteria.
        -- This set is pointed at by SP.  Current_Reference is assigned
        -- the value of the first entry of this set.

        Vocabulary(F:  Open_Files;  SS:  String100;  V:  Vocabulary);

        Combine(S1,S2:  Search_Pointer;  C:  Combination_Modes;
               SP:  OUT Search_Pointer);

        -- Operates on two Search_Set returning a resulting
        -- Search_Set

        Reference(SP:  Current_Reference;  R:  Record_id);

        Get(S:  Search_Pointer;  G:  Inspect_Mode;  SP:  OUT Record_Pointer;
               CR:  INOUT Current_Reference);
        -- Brings a record occurrence into memory and points to it using
        -- a Record_Pointer.  The record occurrence to chose is governed
        -- by the values of the parameters S and G.  The procedure changes
        -- the value of Current_Reference

        Delete(S:  Search_Pointer;  G:Inspect_Mode);

        Save;
        -- A procedure to save a Search_Set
        -- This procedure remains to be defined

        -- Exceptions

END TDB_Search;
```